

C++

пред: 2000 программистов

3 проверки, если отправить заранее. (лучше ко событие твоя)
Если записались и не пришли - штраф.

Защита относительно тестов и ревью

Защита: модификация / доп. реализация
и/или индивидуальный перенос дизайна.

6 мР?

~40 попыток для запуска тестов.

2 недели без

```
#include <stdio.h> - команда препроцессора  
; - не синт. ошибка (даже ошибки не замедляет)  
int main(void): - описание ф-ии main (начало кода)  
{  
    printf("Hello\n");  
    printf; - не вызов функции, копируется  
    return 0;  
}
```

↑ код, который завершается программа

≠ вложенных функций

≠ 1

Исполнение программы начинается с C-runtime.

← тип возвращаемого значения

int main(void)

↑ аргумент = ничего

Си: функция = процедура (void)

лучше всегда писать

Си: int main(void) ≠ int main()

↑
не специфицируем принимаемые аргументы

C++: int main(void) = int main()

printf - вызов функции из стандартной библиотечки (если отсутств. библиотека = ничего)

Внутри printf:

```
printf("Hello \n")
```

перевод строки

" - строка символов

' - код символов

Спец. символы: \...

\n - перевод строки

\t - табуляция

\x - 16-ричный код символа

\\ = \

Таблицы:

- "c:\temp" - попытка открыть файл X
→ "c:\\temp" - ✓
- "c:/temp" - исп. / для разделения путей

Считаем

```
# include <stdio.h>
```

```
int main(void)
```

```
{
```

```
int a, b, c; // обращение к след. аргументу
```

```
scanf("%i %i", &a, &b);
```

```
printf("a+b = %i\n", a+b);
```

```
return 0;
```

```
}
```

printf - быстрее std.out
c.out

%...i

форматки, контролируемые вывод:

? пример %3i = печатать 3 символа. Если нет 3-их, то слева пробелы.

Ошибка компиляции = Error, т.к. синтаксически не явл. корректным.

Warning: предупреждение: возможная ошибка)

```
printf("a + b = %i\n", a + b);
```

чтение из командной строки: warning: типы не совпадают

```
scanf("%i %i", &a, &b)
```

строка и куда читаем
намерения
выделение памяти
создание переменных:

```
int a, b = 0;
```

↑ объявлена
↑ объявлена и инициализирована

! инициализация перед чтением и записью
= затраты памяти.

Локальные переменные выделяются на стеке.

? Стек в **C++**: область памяти для хранения для каждого потока исполнения
+ дешёвое выделение памяти (происходит в начале цикла (при входе в функцию) для локальных переменных).

Переполнение стека на совести программиста:

Stack

- не выделять память в стеке для **биг объектов**
- x глубокая рекурсия
- платформозавис.: расширение стека

Выделение памяти в куче:

- большой V
- один раз (V выносить из цикла)
(без динамического выделения)

Сл: неважно, насколько сложный пре объект,
всё зависит от места, которое сам
определяешь.

Стек: динам. ^{выделение} создание (в цикле) — free

Куча: динам. соз. выд. (в цикле) — дорого
Скipping данных (расположаются глобальные
переменные)


! в куче лучше не исп. глобальные перемен-
ные, кроме констант глобальных.


```
const int g = 25;
```

↑ "в переменную запрещено присваивать"
"после создания значение не изм."

```
constexpr
```

↑ к. времени компиляции — значение, кото-
рое компилятор должен посчитать
в момент компиляции, не в мо-
мент выполнения.

программа  эквивалент программы
оптимизир.
компилятор

Пример: `int a = 2;`
`a = a + 3`  `a = 5`